Overhead Analysis of HTTPS

László Zömbik *

Abstract

Secure web access has remarkable growth. Users would like to exploit the advantages of the Internet for online-banking, for e-commerce or they simply would like to protect their information e.g. with using a secure web mailer. Https is a simple http traffic on top of a security protocol (e.g. SSL, TLS) is used for serving this need. This paper gives detailed analysis of https traffic to aid traffic dimensioning or traffic modelling and even to assist investigation of traffic flow confidentiality.
Keywords: https, SSL, TLS, traffic analysis, traffic flow confidentiality

1 Introduction

The HTTP [5] has the largest traffic on the Internet. Users primarily surf on the Internet to collect public information. Hovewer the number of the web-based commercial services (e-commerce) or private accesses like e-banking rapidly grows. In addition protection of letters using web-based e-mail services has also vital necessity. These services require to protect communication between peers, moreover between users.

Without protecting sensitive web traffic an eavesdropper may deduce parameters or user behavior or an attacker can even identify and impersonate the victim. Therefore, several security solutions have evolved, the two most notable security protocols are the SSL [1] and the TLS [3]. The two protocols are very similar. Before sending user information the client and the server initially negotiate about the security association, including cryptographic algorithms and keys. After the handshake phase protected traffic is transmitted. If a web server uses SSL or TLS then the HTTP [4],[5] communication is secured. In this case the communication is called HTTPS[11].

It is beneficial to have a model for https for bandwidth or for cost estimation. This is extremely important for communications that contain expensive links or the link capacity is limited and secure web access is required. An example can be when the subscribers use satellite terminals, GPRS or UMTS based equipment for browsing on the web.

*Ericsson Hungary, ResearchLab H-1117 Irinyi József utca 4-20. Budapest; Department of Telecommunication and Telematics, Budapest University of Technology and Economics, H-1117, Magyar tudósok körútja 2, Budapest, e-mail: laszlo.zombik@ericsson.com

This paper introduces a methodology for modelling traffic characteristics of security protocols in section 2. Based on this methodology a traffic model for https is presented. In section 3 security protocols of https are shown, section 4 introduces a generic https model. This model is used for the characterisation of https traffic in section 5. Section 6 concerns the user behaviour and the properties of transport and lower layers which can affect the shape of the https traffic. The traffic model is verified by measurements in Section 7.

2 Methodology for modelling traffic characteristic of security protocols

Traffic characteristics of any communication protocol can be modelled easily, since the message format, length and behaviour is known from protocol specifications. However, time dependence and options (ambigously specified behaviour) in the specification can cause statistical properties of traffic characteristics. This kind of statistical properties still can be characterised by traffic measurements. Security protocols unfortunately loses the property of observability due to employment of data confidentality, so from their traffic measurements one hardly can deduce statistical properties. Even most of the security protocols possess the ability of traffic flow confidentality. This not only hides the actual value of data elements, but also the length and the place of information is concealed. In test envinronments all information (e.g. secret material) is aviable for characterising a security protocol, however its behaviour can be quite different than the real-time traffic. Furhermore applied methodology in test envinronments for characterisation of real-time traffic is not manageable. Due to it is impossible to find accountable network administrator (e.g. of an on-line bank) who aggree to disclose confidential information for characterisation.

The following methodology tries to address this issue, it is recommended for investigation of real-time traffic without the need of any confidential information. It requires to know the behaviour of the security protocol, like message flow or where and how cryptographic algorithms, random padding lengths, or compression methods are used.

step 1: message attributes or variables (like port number, length field), used in all type of implementations, should be identified.

step 2: explore which cryptographic and compression algorithms might occur in the security protocol.

step 3: determine or estimate that a specific algorithm in what way affect the message attributes or even the whole message (e.g. an MD5 [6] hash increases the length of message by 16 bytes).

step 4: determine the value of those attributes, which do not depend on the various cryptographic and compression algorithms, but have either fixed length (e.g. Version) or have variable length (e.g. SPI). From the protocol specification or measurements the probability density function of the attributes with variable length can be determined or estimated.

step 5: For those parameters, which depend on some cryptographic or compression algorithms, determine the statistical properties (e.g. the probability of occurence) of a specific algorithm per message attributes. From the protocol specification or measurements (by tracing the list of offered and accepted ciphersuites) this probability can be determined.

step 6: with characterisation of upper and lower layer protocols the effect to the security protocol can be determined. For example an influence from upper layer is the user behaviour. From the lower layer propagation, buffering delays, and other network effects originates. Existing traffic models like HTTP, TCP source models can be used here.

step 7: Model statistical properties of communication, like interarrival time between messages, correlation, etc.

step 8: Simplify the model by neglecting unfrequent messages or rare ciphersuites, or approximating complex messages, and those lenghts.

step 9: Evaluate the model to measurements and feed back the results.

Chapter 4 adresses the first step, the next four steps, and Step 8 are covered in chapter 5. Step 6 is discussed in chapter 6, and the evaluation of the results is discussed in chapter 7.

3 Security protocols for https

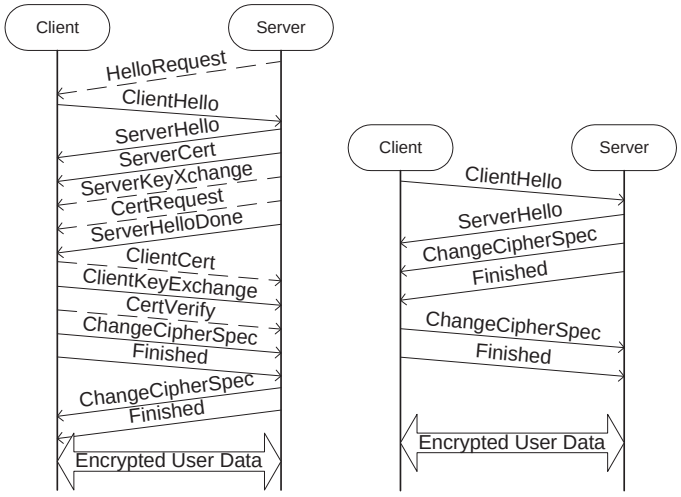
Https can choose between three security protocols. However the oldest one is the Secure Socket Layer version 2 (SSLv2) [2] which contains several security flaws, therefore this is extremely seldom used. Corrected version of this protocol the Secure Socket Layer version 3.0 rapidly became de facto standard. Later the Internet Engineering Task Force (IETF) released an official standard, based on the SSLv3.0, which is called Transport Layer Security (TLS). The difference between the TLS and the SSLv3.0 is minimal: the number of ciphersuites, alert, and certificate types were extended, the method of key calculation was modified and the Handshake finalization message was modified (simplified). The version numbering of TLS shows (version 3.1) that this protocol can be considered as the successor of SSLv3.0. Nowadays secure http communication prefers SSLv3.0, but the newest versions of applications start to use TLS only.

Figure 1 shows the SSL/TLS message flow. Messages with dashed lines are optional. Figure 1(a) shows the full handshake message exchange. The SSL/TLS communication starts with the ClientHello messages, where the client send several ciphersuites to initiate a session. The server select one ciphersuite and answers with a ServerHello message; also certificate of the server and optionally, if the selected ciphersuite requires, serverKeyExchange message is attached. If the server requires client certificate for authentication then this is indicated in the ClientCertRequest message. Finally the ServerHelloDone closes this communication. If a client certificate is requested, then the client attaches its certificate in the answer, also responds with ClientKeyExchange, which contains the keying material. Certificate Verify notification may also be sent by the client. After negotiating and sending the keying

material secure communication should be started. The ChangeCipherSpec message indicates, that the negotiated security procedures must be used right after this message. Therefore, the last message of the client, the Finished is encrypted. If the server receives the ClientKeyExchange, it starts to calculate secret keys. If this is finished the server answers to the client a with ChangeCipherSpec message. The encrypted Finish message of the server closes the handshake. After the handshake phase peers exchange encrypted user data.

If both the server and the client support the Session Key Caching (SKC) mechanisms then after the initial communication they can improve efficiency by reusing existing information. The reduced message flow is presented in figure 1(b).

In the SSL/TLS protocol stack upper layer messages are encapsulated and protected in the lower SSL/TLS Record Layer. There are different methods to achieve confidentiality. *Stream ciphers* have the same length as the plaintext; *Block ciphers* may have padding in order to fill the plaintext to block size of the ciphering algorithm. The padding can be up to 255 bytes in order to provide some data flow confidentiality also. Record Layer messages further contain a generic header (which consists of Type, Version and Length fields) and a Message Authentication Code (MAC).



(a) Full Handshake (b) Session Key Caching

Figure 1: The SSL/TLS message flow.

4 General https model

We denote the user traffic $f(t)$, let θ ($\theta : \Omega \mapsto \Re$) model the interarrival time of messages $f(t) = msg(t)I(\theta < t)$. I(.) is an indicator function, $msg(t)$ is the content of the message at time t.

Furthermore let the encrypted user traffic $f^*(t)$ between the server and the client. The client can initiate multiple sessions, starts with a handshake, therefore let in our model $h(t)$ is the aggregate handshake traffic between the server and the client. The encrypted user traffic is equal to the user traffic fragmented, encapsulated and encrypted to SSL/TLS record layer:

$$f^*(t) = \sum_{i=1}^{N} \Big(RHeader + Enc\Big\{ Compr\{msg_i(t)\} + MAC\big\{ Compr\{msg_i(t)\}\big\}\Big\}\Big)$$

Where + is concatenation, $Enc\{.\}$ means symmetric encryption, and $msg_i(t)$ is the i^{th} fragment of the message $msg(t) = \sum_{i=1}^{N}(msg_i(t))$ such as $N = \lceil \frac{|msg(t)|}{\varphi} \rceil$ and the φ is the size of SSL/TLS fragment.

As SSL/TLS can handle both stream ciphers $S\{.\}$ and block ciphers $B\{.\}$:

$Enc\{.\} = B\{.\}I_B + S\{.\}I_S$

Where I_x indicator function ($I_B I_S = 0$), its value is 1 if the specific cipher is used (else zero). A short definition of block ciphers (let the $\vdash$ symbol be deduction, $|x|$ the length of message x in bytes and plaintext is $\alpha(t)$):

$\beta_B(t) = B\{\alpha(t), key\} = B\{key, \alpha(t) \mapsto \beta_B(t) \big| Pr(\beta_B(t) \vdash \alpha(t)) = 0 \wedge Pr(\beta_B(t) \vdash key) = 0 \wedge |\beta_B(t)| \geq |\alpha(t)|\}$

and Stream ciphers:

$\beta_S(t) = S\{\alpha(t), key\} = B\{key, \alpha(t) \mapsto \beta_S(t) \big| \ |\alpha(t)| = |\beta_S(t)| \wedge Pr(\beta_S(t) \vdash \alpha(t)) = 0 \wedge \ Pr(\beta_S(t) \vdash key) = 0\}$

$Compr\{.\}$ means compression: $\beta_C = Compr\{\alpha(t)\} = \{\alpha(t) \mapsto \beta_C(t) \big| \ |\beta_C(t)| \leq |\alpha(t)|\}$ The *RHeader* is the general SSL/TLS record layer header, it consist of the type, version and length fields. $MAC\{.\}$ is the message authentication code calculation function, this is different in SSL and TLS, but the following definition can be applied to both protocols ($\gamma(t)$ is an arbitrary bitstring):

$\beta_{MAC} = MAC\{\alpha(t), key\} = \{key, \alpha(t) \mapsto \beta_{MAC}(t) \big| \ Pr(\beta_{MAC}(t) \vdash \alpha(t)) = 0 \wedge Pr(\beta_{MAC}(t) \vdash key) = 0 \ \wedge \big(Pr(\alpha(t), key \vdash \gamma(t)) = 0 | \beta_{MAC}(t) \neq \gamma(t)\big)\}$

The generalised https model consists of the encrypted user application data and handshake traffic:

$HTTPS(t) = h(t) + f^*(t)$.

Consider $\underline{\xi}^{(i)} = (\xi_{\lambda_i})$, a vector of probability variable $\underline{\xi}^{(i)} : \Omega \mapsto \Re^i$. The first element refers to the delay of λ_1 message has been arrived relative to a reference time. $\xi_{\lambda_i}, i \neq 1$ are the probability variables of the interarival times of message λ_i and λ_{i+1}.

If no session caching is used then $i \in [1..14]$ and

$$^{(noSKC)}\underline{\lambda}^T = (HelloRequest, ClientHello, ServerHello, ServerCert, ServerKeyExchange, CertRequest, ServerHelloDone, ClientCert,$$

$ClientKeyExchange, ClientCertificateVerify, ClientChangeCipherSpec,$
$ClientFinished$, $ServerChangeCipherSpec$, $ServerFinished$).

If Session caching is set $i \in [1..6]$ and

$$^{(SKC)}\underline{\lambda}^T = (clientHello, ServerHello,\ ServerChangeCipherSpec,$$
$$ServerFinished,\ ClientChangeCipherSpec,\ ClientFinished).$$

The handshake traffic can be divided to client to server $h_c(t)$ and server to client $h_s(t)$ directions. (Figure 1):

$$h_c(t) = \sum_{i \in K} \Big(h_{\lambda_i} I(\xi_{\lambda_i} < t) \Big);\ h_s(t) = \sum_{i \in L} \Big(h_{\lambda_i} I(\xi_{\lambda_i} < t) \Big);$$

where $K_{SKC} = \{1, 5, 6\}$ and $L_{SKC} = \{2, 3, 4\}$ at session reuse, and $K_{noSKC} = \{2, 8, 9, 10, 11, 12\}$ and $L_{noSKC} = \{1, 3, 4, 5, 6, 7, 13, 14\}$ when https not uses previous session information. h_{λ_i} is the handshake message λ_i, $I(.)$ is an indicator function, its value is set when the corresponding message is sent.

4.1 Format of Handshake Messages

In this section the formal description of the handshake messages will be introduced. The *RHeader* is the general SSL/TLS record layer header, it consist of the type, version and length fields, while the *HHeader* is the general SSL/TLS handshake header, it contains the handshake type, and length fields. While *Ver* is the version attribute of the sender. I_x is a generic indicator function, its value one if and only if message x exists, otherwise zero.

The Record layer adds the record layer header to all upper layer protocol (handshake, application data) messages. If a message is supposed to be encrypted, then after fragmentation the $\Psi(.) = RHeader + Enc\{Compr\{.\} + MAC\{Compr\{.\}\}\}$ transformation is applied to it.

HelloRequest consist only of the handshake layer header $h_{helloReq} = HHeader$.

ClientHello contains *HHeader*, *Ver*, and a *Rand* random data.

$h_{clienthello}(t) = HHeader + Ver + Rand + CSessionID + CCipherSuites + Compression$

Where $CSessionID, CCipherSuites, Compression$ attributes are the session ID of the client (variable length random ID), list of ciphersuites and list of compression methods offered by the client. In every implemetation those fields additionaly contain length fields even if this is not specified in the SSL/TLS standards. The server answers to the clientHello message, choosing ciphersuite and compression method:

$h_{serverhello}(t) = HHeader + Ver + Rand + SSessionID + SCipherSuite + Compression$

The server certificate message contains a server specific certificate or certificate list. CertificatesLength indicates the length of the certificate, this variable is not in the SSL/TLS specification.

$h_{SCert}(t) = HHeader + CertificatesLength + CertList(server)$

In the Server Key Exchange message if RSA key transport is choosen in the ciphersuite, RSA parameters, if Diffie Hellman key exchange is selected, then DH parameters are sent. The message should also contain a signature to authenticate the traffic:

$h_{SKeyXch}(t) = HHeader + Ver + I_{RSA}RSA + I_{DHE}DH + I_{Sig}Sig$, where

$I_{RSA}I_{DHE} = 0$. DH contains the Diffie-Hellmann public variables, g, n and Y_s, RSA [7] contains the modulus and the exponent. I_{Sig} indicates whether a signature exists, Sig is a signature algorithm, specified in the SCipherSuite. If DSS [8] authentication exists, an SHA-1 [9] hash, if the signature is based on RSA, then an MD5+SHA1 hash is applied:

$Sig = I_{RSA}(MD5 + SHA) + I_{DSS}(SHA)$

The server can optionally ask for client certficate for mutual authentication. In this case the known certificate types (ClientCertTypes) and the known Cerificate authority names (CAname) are specified. In addition implementations also contain one byte long attribute for indicating the number of known certificate types (CertTypesCount).

$h_{CertReq}(t) = HHeader + CertTypesCount + ClientCertTypes + CAname$

The ServerHelloDone consists of:

$h_{SHelloDone}(t) = HHeader$

Client certificate is similar to the server certificate message, however it contains the Client certificate or certificate list:

$h_{CCert}(t) = HHeader + CertificatesLength + CertList(client)$

Client Key exchange message: if RSA key transport is selected contains Version and radom number encrypted by the public RSA key found in the certificate. If DH method is selected the public DH value is attached:

$h_{CKeyXch}(t) = HHeader + I_{RSA}\{Ver + RND\}RSA_{Pub} + I_{DHE}Y_{sDH}$

The optional Client Certificate Verify message includes either DSS or RSA signature:

$h_{ClCertVerify}(t) = HHeader + I_{Sig}Sig$,

The handshake phase is closed by the Finished message, generally this message is encrypted by the negotiated algorithms. This message contains verification data consists of 36 bytes in case of SSL and 12 bytes in TLS.

$h_{Finished}(t) = HHeader + VeryfyData$

In SSL/TLS there are two additional non handshake messages, the Alert and Change CipherSpec message. The former indicates warning or fatal error, while the latter, indicates, when the negotiated algorithms should be used. (Both can be in encrypted form):

$h_{Alert}(t) = AlertLevel + AlertDescription$ and $h_{ChCipherSpec}(t) = CipherSpecType$

5 Modelled traffic characteristics of https

In this chapter Step 2,3,4,5 and 8 of the methodology for modelling traffic characteristic of security protocols are shown.

5.1 Cryptographic algorithms

In the following *ciphers used in SSL/TLS implementations* are shown: RSA is used for Authentication and Key transport, but also DH or DHE is used for key generation and DSS for authentication. For symmetric encryption 3DES, DES, RC4 and IDEA is used, and MD5 and SHA-1 for message integrity.

Blocksize or output length of the used cryptographic algorithms are the following: MD5 uses 16, SHA-1 uses 20 bytes. Block ciphering algorithms like DES, 3DES, IDEA have 8 bytes blocklength, RSA generates as long ciphertext as the size of modulus [10] so 64, 128, 256 bytes (if the key length is 512, 1024, 2048 bits). Output length of RC2, RC4 algorithms is the same as the length of the input plaintext.

Size of the cryptographic parameters that can be sent between the client and the server are various. The modulus of RSA can be around 64, 128, 256 bytes (512, 1024, 2048 bits), however the RSA exponent is usually 3 bytes. The length of Diffie-Hellmann generator is usually 1 bytes, while the modulus and Y_s typically have 64, 96, 128 bytes (512, 768, 1024 bits) length. A DSA signature (containing the r and the s) requires 40 bytes, while RSA signature requires at least 64 or 128 bytes (if key length is 512, 1024 bits respectively).

5.2 Length of Message Variables

Constant length variables of the SSL/TLS messages and their sizes are the following: RHeader 5, HHeader 4, Ver 2, Rand 32, CertificatesLength 3, CertTypeCount , AlertLevel, AlertDesc, ChCipherSpecType and ComprMethod 1 bytes.

Variable length message attribute is only the client SessionID data, whose length can be between 0 and 32 bytes, but in our measurement it is either 0 or 32.

Variable length message attributes which depend on some security properties are the following. The CipherSuiteList variable of ClientHello contains between 5 and 24 ciphersuits however, in most cases 8 and 11 ciphersuits are sent to the server (one half of the samples contained 11 and one quarter of the samples had 8 ciphersuite). Since one ciphersuit consumes 2 bytes, the length Ciphersuite variable can be modelled as: $Pr(x < 0.5)22 + Pr(0.5 < x < 0.75)16 + Pr(x > 0.75)Unif(10, 48)$.

The length of the server and the client certificate depends on several parameters, for example the number of X.509 extensions and signature algorithm. In addition the lenght of certificate depends on the length of the bounded public key and signature. Public keys used on the Internet have 512, 1024 or 2048 bits length, however the latter is not typical yet. Therefore the length of the Certificate payload is modelled as uniform distribution between 600 and 1000 bytes.

The RSA public modulus has the same length as the keysize, so it can be 64, 128 bytes. The RSA public exponent usually has 3 bytes length. Diffie-Hellman generator is usually 1 byte, while the modulus and Y_s can be 64, 96, 128 bytes. The length of Signature data (Sig) can be 0, (in case of no signature) 40, (if the signature algorithm is DSS) or 64, 128 (in case of 512 or 1024 bits length RSA) bytes.

The Client *CertTypesCount* in the Certificate Request can be between 1 and 255 bytes, but in our measurement this is between 30 and 200. Therefore let the length of ClientCertTypes is modelled as uniform distribution between 30 and 200 bytes.

The RSA encrypted message attribute in the Client Key Exchange message ($\{Ver + RND\}RSA_{Pub}$) has the same length as the RSA modulus size so 64 or 128 bytes. In the Finished message the Verification data is either 36 bytes if SSL is used or 12 bytes in case of TLS.

5.3 Length of Messages

The traffic characteristic of https can be defined using the generalised https model:

$|HTTPS(t)| = |h(t)| + |f^*(t)|$,

Handshake messages can be divided to client and server communication,

$|h(t)| = |h_c(t)| + |h_s(t)|$, where

$|h_c(t)| = \sum_{i \in K} \Big(|h_{\lambda_i}| I(\xi_{\lambda_i} < t)\Big)$; $|h_s(t)| = \sum_{i \in L} \Big(|h_{\lambda_i}| I(\xi_{\lambda_i} < t)\Big)$;

Let $\underline{\phi}^{(i)} = (\phi_{\lambda_i})$, $\underline{\phi}^{(i)} : \Omega \mapsto \Re^i$ probability variable for the length of the λ_i message. In this case the handshake traffic length:

$$|h_c(t)| = \sum_{i \in K} \Big(\phi_{\lambda_i} I(\xi_{\lambda_i} < t)\Big); \; |h_s(t)| = \sum_{i \in L} \Big(\phi_{\lambda_i} I(\xi_{\lambda_i} < t)\Big);$$

Furthermore let the probability density function of ξ_{λ_i}: $f_{\lambda_i}(x)$.Suppose that $f_{\lambda_i}(x)$ is time invariant. Note that if $f_{\lambda_i}(x)$ and the distribution of ϕ_{λ_i} is known then the handshake traffic can be modelled. In the following several models for $\phi_{\lambda_i}(x)$ and for its distribution is presented, while models for $f_{\lambda_i}(x)$ is discussed in chapter 7.2.

The length of Record and Handshake layer header is fixed, 5 and 4 bytes respectively ($|RHeader| = 5, |HHeader| = 4$). Let the ($\{a, b, c\}$) notation is a specific distribution between the set of a,b and c. The general model of Table 1 shows the length of handshake messages using parameter estimation from Section 5.2.

Record layer encryption is used to protect user data, finish and alert messages.

$|f^*(t)| = \sum_{i=1}^{N} \Big(|RHeader| + |B\{payload\}| I_B + |S\{payload\}| I_S\Big)$,

where the payload before encryption:

$payload = Compr\{msg_i(t)\} + MAC\{Compr\{msg_i(t)\}\}$.

The MAC algorithm is calculated as the outmost function is a hash. Therefore the length of MAC depends on the hash algorithms: $|MAC\{.\}| = I_{MAC}(16 I_{MD5} + 20 I_{SHA})$

The block ciphered data length:

$|B\{payload\}| = |Compr\{msg_i(t)\}| + |MAC\{Compr\{msg_i(t)\}\}| + |Padding|$

Let the blocklength is k bit. If the communication uses no traffic flow confidentiality (so the padding fills out only the last block), then the padding length can be modelled as uniform distribution: $|Padding| = Unif(0, k/8)$. Otherwise if traffic

flow confidentiality is set then the padding can fill at least the last data block, but its length can reach 255 bytes:

$i \cdot blocksize + Unif(0, k/8) \leq 255$, where i is a probability variable, such $i \in \aleph$ and its minimal value is $Univ(0, k/8)$ and maximal value is 255. The distribution of this probability variable can be modeled with uniform distribution $i = Unif(Unif(0, k/8), 255)$.

For example, blockciphers with blocksize of 64 bits and with no traffic flow confidentality, $|Padding| = Unif(0, 8)$. This is the common case in our measurements.

The stream ciphered data length:

$|B\{payload\}| = |Compr\{msg_i(t)\}| + |MAC\{Compr\{msg_i(t)\}\}|$

5.4 Simplifications in the model

In the *First refinement model* we made several improvement to adjust our model to the common properties of the implementations. Therefore the size of the ClientHello and ServerHello message is increased by four and one respectively. The implementations uses extra length fields in those messages. The Certificate of the client and the server is modelled by an uniform distribution between 600 and 1000 bytes, because the length of all the certificates were between those values.

The *Second refinement model* contains simplifications of various algorithms. In all measurements no compression is used by implemetations, so

$|Compr\{f(t)\}| = |f(t)|$.

In our measurements $RSA\_RC4\_128\_MD5$ was the most frequently used ciphersuite (89.5%), due in most implementation it is included, and it has the highest preference. Occurence of $RSA * \_RC4\_56\_MD5$ is 1.7%, $RSA\_RC4\_128\_SHA$ is 0.35%, $RSA * \_RC4\_56\_SHA$ is 8.2%. The remaining block ciphers, (like $DHE\_RSA\_3DES\_SHA$) are used only in 0.25% of the samples. Therefore we can conclude, that in our measurements the stream ciphers have dominance(99,75%), therefore let us simplify our model as $I_B = 0, I_S = 1$.

In most cases (99.8%) RSA is used, so let us simplify our model further, as $I_{RSA} = 1, I_{DSS} = 0, I_{DH} = 0, I_{DHE} = 0$. From the RSA ciphersuits 1.8% was RSA_EXPORT (with 512 bits), 8.2% was RSA_EXPORT_1024 (with 1024 bits) and 90% of the ciphersuits was RSA with no limitation. However unlimited RSA ciphers used only 1024 bits, therefore the 1024 bit lenght RSA had the dominance (98.2%). SHA algoithm is used 8.8% of the https connection, while MD5 is commonly used 91.2%. All communication used integrity check, so $I_{MAC} = 1$.

SSL is used in 52% of the https traffic (TLS is used in 48% of communication). Therefore we simplify that the length of finished message is (with 50% probability) either 45 or 21 bytes. In ServerKeyExchange and in ClientVerify messages SHA signature is used.

The SessionID in the clientHello is set when an SSL/TLS session has been set up previously, and no new SSL/TLS connection is forced. Therefore the usage of SessionID is set the case of session caching.

The modified length of handshake messages can be found in Table 1.

Note that the calculation of the record layer length is simplified ($N = \lceil \frac{|msg(t)|}{\varphi} \rceil$):

$$|f^*(t)| = \sum_{i=1}^{N} \Big(|RHeader| + |msg_i(t)| + I_{MD5}|MAC\{msg_i(t)|\} \Big) =$$
$$\sum_{i=1}^{N} \Big(|RHeader| + I_{MD5}|MAC\{msg_i(t)|\} \Big) + |f(t)| = 21N + |f(t)|.$$

In the *Third refinement model* unfrequent messsages (which occurence has under 0.5%) are removed (Table 1). Since all optional handshake messages are removed, only the ClientHello, ServerHello, SCert, SHelloDone, ClKeyExchange, ChCipherSpec, Finished messages remain.

The *Session Key Caching* technique requires less messages, the length of those messages can be found in Table 1.

6 Behaviour of User and Network

The nature of the https traffic highly depends on user behaviour. However, https services, like internet banking, e-commerce, secure mailing, etc. delimits the behaviour of the user. Therefore three different types of secure http service can be easily separated. In the first type the user interactively uses the service, reads and fills https forms and sends them back (e.g. a questionaire). In the second type the user primarily reads information, and seldom sends data (e.g. querying an on-line bank). The third type of secure web service is a secure mailer, where the user reads and sends mails. In this case the user can attach a large file for uploading, or the user may also download large files.

The user generared traffic is correlated to the handshake traffic. Every new standalone HTTP object download preceeds a new handshake message flow.

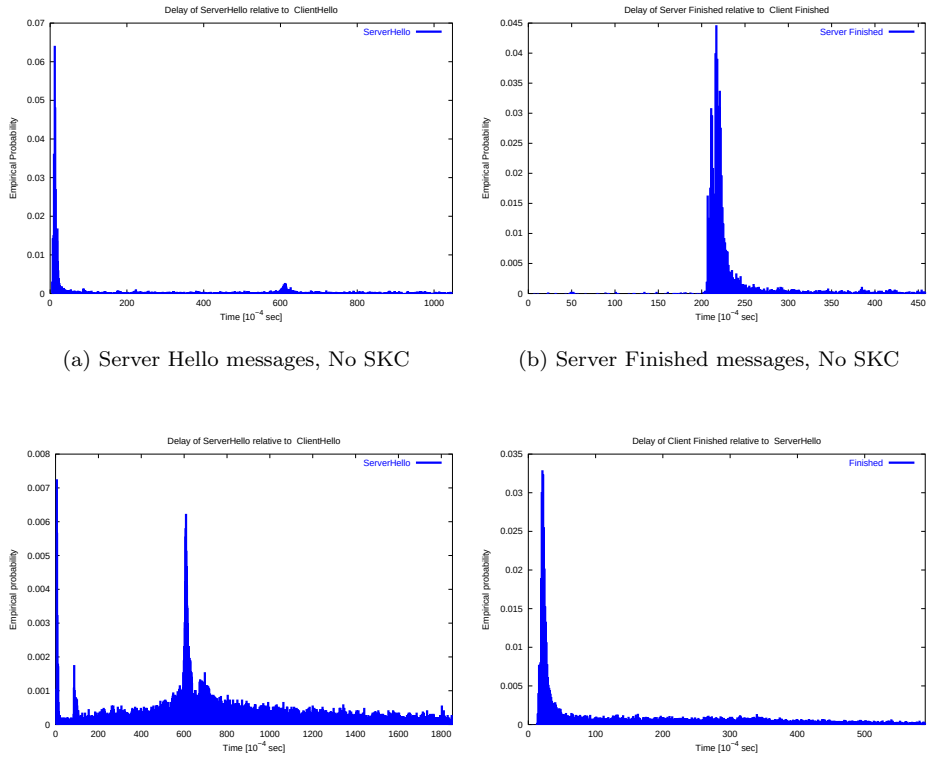
The secure layer opens a new secure connections for each new objects. Therefore if the session caching is not set, then the https traffic:

$$HTTPS(t) = \sum_{j=1}^{\#objects} \sum_{i=1}^{N} \Big(h_{noSKC,i,j}(t) + RHeader_{i,j} + msg_{i,j}(t) + padding_{i,j} + MAC_{i,j} \Big).$$

In case of session key caching is used, the first object download uses full handshake (because there is no preceeding shared knowledge), however further objects uses session cacing mechanisms:

$$HTTPS(t) = h_{noSKC,1}(t) + \sum_{i=1}^{N} (RHeader_i + msg_{i,1}(t) + padding_{i,1} + MAC_{i,1} +$$
$$\sum_{j=2}^{\#objects} \sum_{i=1}^{N} \Big(h_{SKC,i,j}(t) + RHeader_{i,j} + msg_{i,j}(t) + padding_{i,j} + MAC_{i,j} \Big).$$

The network also can distorse the traffic shape of https; buffering delays, network congestion, the state of TCP (e.g. in slow start), propagation delays may cause significant deviation.



(a) Server Hello messages, No SKC

(b) Server Finished messages, No SKC

(c) Server Hello messages, SKC

(d) Finished messages, SKC

Figure 2: Empirical distributions of interarrival times

However in our study the delay is corrected by the actual RTT. Therefore effects from the network layer can be ignored.

7 Results

Our measurements has been collected in two scenarios. In the first configuration we monitored 100Mbps backbone line. The second measurements was collected on the 10Mbps network segments where https server located. In both scenarios we monitored real https traffic, which sent on TCP port 443. In the first configuration more than 120000, while in the second configuration more than 20000 https communication (containing handsake and user data traffic) were collected.

7.1 Length of HS messages

We have four methods to estimate the length of handshake messages. The first one is originated from the general traffic characteristics in Section 5.3. In this model there are several unresolved variables the first refinement model resolves. This might gives a rough estimation, but if it is not sufficient enough, then the second refinement model can be used, where the unfrequent variables are removed. In the third refinement model unfrequent messages are also neglected. The SKC model can only be used for those connections uses session key caching mechanism (Table 1).

7.2 Time dependence

The lower layer of the SSL/TLS protocols optimises the handshake performance by groupping successive handshake messages. The empirical density of the interarrival times of the groupped messages are shown in Figure 2. The distortion caused by network is corrected by the actual RTT, furthermore the data was collected in the second scenario, in the same LAN the server resides. Figure 2(a) demonstrate that the server immediately (2-3ms) responds to ClientHello. In some cases however the server looks up its database for the client. This causes extra delay (60ms) which is signifficant in the case of session caching Figure 2(c).

The delay between the two finished messages (20ms) in non-session key caching case (Figure 2(b)) is signifficant compared to session caching case. The cause of this difference can be found in the key agreement mehanisms, since the server processes the CPU intensive ClientKeyExchange message, while in the session key caching case this step is replaced to a faster one.

Note that the empirical probability density functions can be used for estimating the density funtion of ξ_{λ_i}.

7.3 Overhead of https

Handshake overhead

The handshake overhead is defined as the ratio of the handshake and the total traffic of a connection.

$$R_{handsake} = \frac{|h(t)|}{|h(t)| + |f^*(t)|} = \frac{|h(t)|}{|h(t)| + N(|RHead| + |padding| + |MAC(f(t))|) + |f(t)|} = \frac{|h(t)|}{|h(t)| + N(|RHead| + |padding| + |MAC(f(t))|) + N\varphi - \nu},$$

where $\nu = 0..(\varphi - 1)$.

The maximal value of handshake overhead (when no padding, MAC):

$R_{handsake_{MAX}} = \frac{|h(t)|}{|h(t)| + 5N + |f(t)|} = \frac{|h(t)|}{|h(t)| + 6}$

Handshake messages (λ_i)	Empirical	general model ϕ_{λ_i}	refinement model1	refinement model2	refinement model3	SKC
Hello Req*	9	9	-	9	-	-
Client Hello*	$p = 0.5 : 102$ $p = 0.25 : 96$ $p = 0.25 : Unif(45, 110)$	$44 + 32I_{sessionID} + 2(\#Ciphersuite)$	$48 + 32I_{SKC} + 2(\{8, 11\})$	({ 96, 102 })	({ 96, 102 })	({ 96, 102 })
Server Hello	74	73	74	74	74	74
Server Cert	Unif(600,1000)	$\phi_{cert} = \phi_{cert}(server)$	12+ Unif(600, 1000)	806	806	-
Server Key Exchange	210	$11 + I_{RSA}(3 + (\{64, 128\})) + I_{DHE}(1 + 2(\{64, 96, 128\})) + I_{Sig}((\{40, 64, 128\}))$	-	180	-	-
S Hello Done	4	4	-	4	4	-
Client Cert	Unif(600,1000)	$\phi_{cert} = \phi_{cert}(client)$	12+ Unif(600, 1000)	806	806	-
Client Key Exch	127	$9 + I_{RSA}(3 + (\{64, 128\})) + I_{DHE}(1 + (\{64, 96, 128\}))$	-	140	140	-
Cert Verify	37	$9 + I_{Sig}(\{40, 64, 128\})$		49	-	-
ChCipherS*	6	6	-	6	6	6
Finished*	({45, 21})	$9 + (\{36, 12\})$	-	({45, 21})	-	-
Alert*	7	7	-	7	7	7
Encrypted Finished*	p=0.067: 37 p=0.015: 41 p=0.869: 61 p=0.047: 65	$\{21, 45\} + Unif(0, 8) + \|MAC\|$	-	({37, 41, 61, 65})	({37, 41, 61, 65})	61
Encrypted Alert*	p=0.802: 23 p=0.191: 27	$7 + Unif(0, 8) + \|MAC\|$	-	23	23	23
Record Layer	-	$\|Compr\{msg_i(t)\}\| + Unif(Unif(0, 8), 255) + \|MAC\{Compr\{msg_i(t)\}\}\|$		21+ $\|f(t)\|$	21+ $\|f(t)\|$	NA

Table 1: Length of handshake messages. * length incudes the Record layer encapsulation

	Session Caching			Non Session Caching		
Handshake traffic	Min	Max	Typical	Min	Max	Typical
Client to Server	125	261	169	159	369	264
Server to Client	122	150	146	$131 + cert$	$369 + cert$	$155 + cert$
Overhead						
Client to Server	$\frac{125}{125+N(33+\varphi)}$	0.9775	0.1392*	$\frac{159}{159+N(33+\varphi)}$	0.984	0.201*
Server to Client	$\frac{122}{122+N(33+\varphi)}$	0.9615	0.1225*	$\frac{131+cert}{131+cert+N(33+\varphi)}$	$\frac{1-6}{375+cert}$	$\frac{1-1045}{1200+cert}$ *

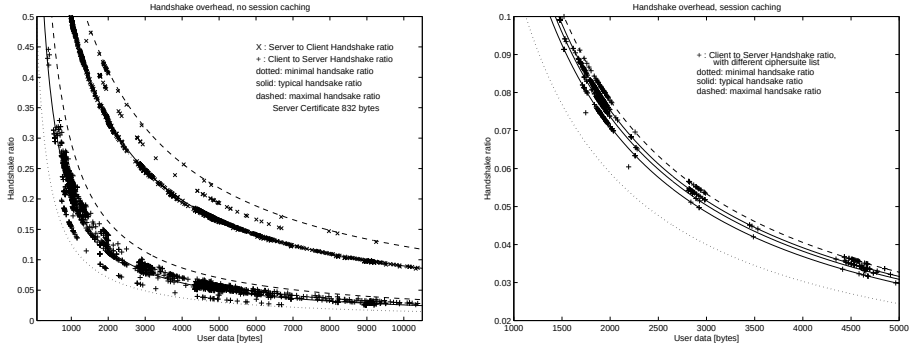
Table 2: Length of handshake messages and overhead. * for 1kbyte user data

when N=1, $\nu = \varphi - 1$. The minimal value of the handshake overhead (blockcipher with 64 blocklength, SHA):

$R_{handsake_{MIN}} = \frac{|h(t)|}{|h(t)| + |f(t)| + 33N} = \frac{|h(t)|}{|h(t)| + N(33+\varphi)}$.

The most possible overhead value (the ciphersuite TLS_RSA_RC4_MD5):

$R_{handsake} = \frac{|h(t)|}{|h(t)| + |f(t)| + 21N} = \frac{|h(t)|}{|h(t)| + N(21+\varphi) - \nu}$. Table 2 shows the maximal and minimal values of handshake communication using the models in Table 1.



(a) No session caching

(b) Session Caching

Figure 3: Handshake ratio (empirical)

Figure 3 shows how the handshake ratio depends on the user traffic. The dotted and dashed lines are the minimal and maximal borderlines of the handshake ratio. In figure 3(a) the server to client and the opposite direction of handshake ratio is represented. Figure 3(b) proves that even the ciphersuite list affects the handshake ratio, in the figure the ClientHello contained 3,5,6,8,11 ciphersuite, thus the handshake ratio increases.

Cost for security The traffic cost for securing http is defined as the ratio of the sum of the SSL/TLS control messages, record layer overhead and the user traffic.

$$\eta = \frac{|h(t)| + N(|RHead| + |MAC(f(t))| + |padding|)}{|f(t)|} = \frac{|h(t)| + N(|RHead| + |MAC(f(t))| + |padding|)}{N\varphi - \nu},$$

The miminimal cost line (when no padding and no MAC)
$\eta_{min} = \frac{|h_{min}(t)|}{N\varphi} + \frac{5}{\varphi}$,
The maximal cost line (blockcipher, SHA MAC, etc.)
$\eta_{max} = \frac{|h(t)_{max}| + 33N}{|f(t)|} = |h(t)_{max}| + 33$

It is obvious that the cost function depends on the user traffic. If the user can estimate the amount of traffic he sends [13], the additional network traffic can be determined. Therefore this information can be the basis of the network dimensioning, or bandwidth estimation.

8 Conclusion

Https model can be used not only for describing secure web traffic, but it can be used for characterisation of secure IMAP, and secure POP3 [12] also. User traffic and network congestion can distorse the shape of https, however the user behaviour is determined by the secure service. In this paper a general https model, and an exact traffic model together with simplified traffic models are introduced.

References

[1] Frier, Karlton, Kocher, "The SSL 3.0 Protocol", Netscape Corp., 1996. Internet Draft, Work in Progress

[2] Kipp E. B. Hickman, "The SSL Protocol" Netscape Corp., 1995. Internet Draft, Work in Progress

[3] Dierks, T. and C. Allen, "The TLS Protocol", RFC 2246, 1999. Proposed Standard

[4] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, May 1996.

[5] Fielding et Al, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, 1999.

[6] R. Rivest, "The MD5 Message-Digest Algorithm", RFC1321, 1992.

[7] Bruce Schneier, "Applied Cryptography" 2nd Edition, John Wiley, 1996.

[8] FIPS PUB 186-2, National Institute of Standards and Technology, 2000.

[9] FIPS PUB 186-1, National Institute of Standards and Technology, 1995.

[10] Public Key Cryptography Standards PKCS#1, RSA laboratories, 1998.

[11] E. Rescorla, "HTTP Over TLS", RFC2818, 2000.

[12] C. Newman, "Using TLS with IMAP, POP3 and ACAP", RFC2595, 1999.

[13] P. Vaderna, E. Strömberg, T. Éltetô, "Modelling the Performance of HTTP/1.1", submitted to GLOBECOM, 2004.